# Scala for Professionals

## A Definitive Tour of Advanced Scala Programming

# Agenda

- Short Recap of Important Scala Basics
- Object Functional Programming in Depth
- Fundamentals of Implicits
- Mastering the Type System
- Implicits & the Type System
- Custom Control Structures
- Custom Scala Collections
- Parallelism, Concurrency, and Stream Processing
- *Advanced Functional Programming with scalaz*[*]
- *A Brief Tour of Macro & Metaprogramming*[*]

---

[*] Optional sections, time and schedule permitting

# What's This All About?

- Introducing you to the knowledge & features of Scala necessary to act as a Lead or Architect level developer
- Provide you hands on experience using these features, so that you're comfortable working with them *and* explaining them to others

# Case Study: "AirScala"

# Case Study: “AirScala”

- This course is very exercise heavy: the best way to absorb new concepts is to write code with them.
- The case study for this project is “AirScala”: A Scala based system for Airline Reservations and Scheduling.
  - The code we build will accomplish tasks such as reconciling travel itinerary requests with flight availability.

# Short Recap of Important Scala Basics

# Object Oriented Programming Basics

- What's the difference between a field and a class parameter?
- What's a singleton object? What are singleton objects used for?
- What are the additional features of case classes?
- Describe the Uniform Access Principle (UAP), and how it is implemented in Scala.

# Collections & Functional Programming

- How do we create an instance of a collection?
  - How does that work behind the scenes?
- How can we mutate an immutable collection?
- What's a function?
  - What's a Higher Order Function?

# Collections & Functional Programming

- Describe three important Higher Order Functions of Collections:
  - What's the purpose?
  - What does the signature look like?
  - Give an example.

# For-expressions and for-loops

- What's the difference between for-expressions and for-loops?
- Describe generators, filters, and definitions.
- To which types can we apply for-expressions?

# Inheritance

- What's the difference between overriding and defining an abstract member?
- Can we override a `def` with a `val`?
  - What about the other way around?

# Traits

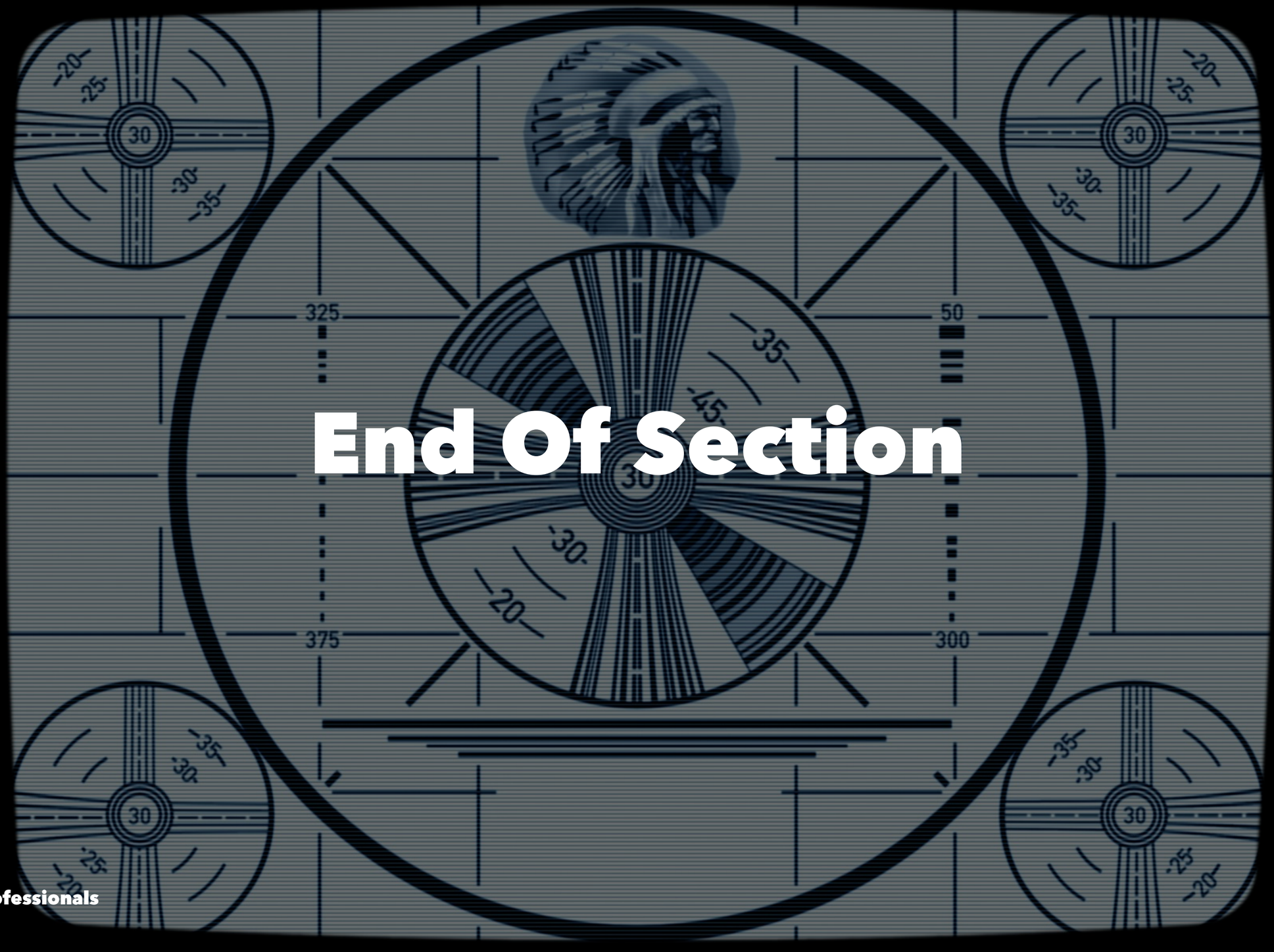
- What's the effect of applying the `sealed` modifier?
  - What's the purpose?
- How can we mix a trait into a class?
- What's the difference between multiple inheritance and mix-in composition using traits?

# Pattern Matching

- What does the syntax for a `match` expression look like?
- Describe some of the available patterns.
- What's a pattern guard?
  - Why is it useful?

# Dealing With Optional Values

- What's the idiomatic way to represent optional values in Scala?
- Describe two alternatives for handling optional values in Scala.



# End Of Section